FAT-CAT: Frequent Attributes Tree Based Classification

Jeroen De Knijf

Universiteit Utrecht, Department of Information and Computing Sciences
PO Box 80.089, 3508 TB Utrecht
The Netherlands
jknijf@cs.uu.nl

Abstract. The natural representation of XML data is to use the underlying tree structure of the data. When analyzing these trees we are ensured that no structural information is lost. These tree structures can be efficiently analyzed due to the existence of frequent pattern mining algorithms that works directly on tree structured data. In this work we describe a classification method for XML data based on frequent attribute trees. From these frequent patterns we select so called emerging patterns, and use these as binary features in a decision tree algorithm. The experimental results show that combining emerging attribute tree patterns with standard classification methods, is a promising combination to tackle the classification of XML documents.

1 Introduction

In recent years there has been a growing interest from the knowledge discovery and data mining community in analyzing and mining XML data. The main reasons for this are the growing amount of semi-structured data and the widespread adoption and use of XML as the standard for semi-structured data. Frequent tree mining is a data mining technique that is able to exploit the information on the structure of the data present in XML-databases. Frequent tree mining is an instance of frequent pattern mining, specialized on tree structured data. Some well known algorithms are described in [2,13,14]. Briefly, given a set of tree data, the problem is to find all subtrees that satisfy the minimum support constraint, that is, all subtrees that occur in at least $n\%$ of the data records.

Classification is a major theme in data mining; the goal of classification is to predict the class of objects based on their attributes. Within the frequent pattern mining paradigm different classification approaches have been developed. In the work of Li et. al [11] classification is based on association rules i.e., it computes frequently occurring items associated with a class label. A drawback of this approach when applied to XML data is that the structure of XML data is lost. Frequent pattern based classification techniques that are suited for structured data are described in the work of Zaki and Aggarwal [15], Geamsakul et al. [9] and Bringmann and Zimmermann [5]. The first method computes frequent trees, orders these based upon some rule strength measures, and uses a decision list

N. Fuhr, M. Lalmas, and A. Trotman (Eds.): INEX 2006, LNCS 4518, pp. 485–496, 2007.

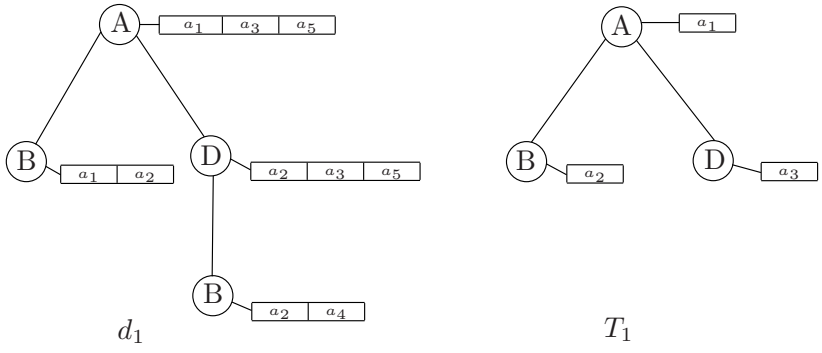

Fig. 1. T_1 is an induced subtree of the rooted ordered attribute tree d_1

approach to build a classifier. The other two methods compute interesting tree or graph patterns and use these as binary features in a standard classification method such as decision trees [12].

A drawback of the current frequent pattern based classification algorithms is that the existence of attributes associated with structured data is ignored, and hence potentially useful information is neglected. In previous work [10] we have described FAT-miner an efficient algorithm to mine tree structured data where attributes play an important role in the mining process. In this work we describe FAT-CAT; a classification method for XML data, based on frequent attribute trees. We applied this classification method to the Wikipedia [7] classification dataset, and discuss the results. Furthermore, these results are compared with those of a frequent pattern approach, that ignores the attributes of the dataset.

2 The Mining Algorithm

In this section we provide the basic concepts and notation used in this paper and describe the mining algorithm.

2.1 Preliminaries

A labeled rooted ordered attribute tree $T = \{V, E, \leq, L, v_0, M\}$ is an acyclic directed connected graph which contains a set of nodes V, and an edge set E. The labeling function L is defined as $L : V \rightarrow \Sigma$, i.e., L assigns labels from alphabet Σ to nodes in V. The special node v_0 is called the root of the tree. If $(u, v) \in E$ then u is the parent of v and v is a child of u. For a node v, any node u on the path from the root node to v is called an ancestor of v. If u is an ancestor of v then v is called a descendant of u. Furthermore there is a binary relation '$\leq$' $\subset V^2$ that represents an ordering among siblings. The size of a tree is defined as the number of nodes it contains; we refer to a tree of size k as a k-tree. The set of attributes is denoted by : $\mathcal{A} = \{a_1, \ldots, a_n\}$, where each attribute takes its

value from a finite domain. We further assume that there is an ordering among the attributes; i.e., $a_j \prec a_k$. To each node v in V, a non-empty subset of $\mathcal{A}$ is assigned; we call this set the attributes of v. More formally: $M : V \rightarrow \mathcal{P}(\mathcal{A}) \backslash \{\varnothing\}$.

Given two labeled rooted attribute trees T_1 and T_2 we call T_2 an induced subtree of T_1 and T_1 an induced supertree of T_2, denoted by $T_2 \preceq T_1$, if there exists an injective matching function Φ of V_{T_2} into V_{T_1} satisfying the following conditions for any $v, v_1, v_2 \in V_{T_2}$:

1. Φ preserves the labels: $L_{T_2}(v) = L_{T_1}(\Phi(v))$.
2. Φ preserves the order among the siblings: if $v_1 \leq_{T_2} v_2$ then $\Phi(v_1) \leq_{T_1} \Phi(v_2)$.
3. Φ preserves the parent-child relation: $(v_1, v_2) \in E_{T_2}$ iff $(\Phi(v_1), \Phi(v_2)) \in E_{T_1}$.
4. Φ preserves the attributes: $\forall v \in V_{T_2} : M(v) \subseteq M(\Phi(v))$.

In figure 1 an example data tree (d_1) is shown together with one of it's subtrees. Let $D = \{d_1, \ldots, d_m\}$ denote a database where each record $d_i \in D$, is a labeled rooted ordered attribute tree. Let $C = \{c_1, \ldots, c_k\}$ be the k classes in the data. With D_{c_i} we denote the set of records in the database that has class label c_i, likewise with $D_{\overline{c_i}}$ the set of records in the database is denoted that has a class label different form c_i. For a given labeled rooted ordered attribute tree T, we say T occurs in a transaction d_i if T is a subtree of d_i. Let $\sigma_{d_i}(T) = 1$ if $T \preceq d_i$ and 0 otherwise. The support of a tree T in the database D is then defined as $\psi(T) = \sum_{d \in D} \sigma_d(T)$, that is the number of records in which T occurs one or more times. Likewise, the support within a class c_i of a tree T is defined as $\psi(T|c_i) = \sum_{d \in D_{c_i}} \sigma_d(T)$, that is the number of records with class label c_i in which T occurs one or more times. T is called frequent within the class c_i if $\psi(T|c_i)/|D_{c_i}|$ is greater than or equal to a user defined minimum support ($minsup$) value.

In general, the goal of frequent tree mining algorithms is to find all frequently occurring subtrees in a database. In the current setting, we are interested in all frequently occurring subtrees within the classes. Besides the frequency within a class constraint, an additional requirement is that a pattern is an emerging pattern for its class, i.e. patterns that occur often in one class and rarely in any other class. These patterns can have very discriminating properties, which is useful for classification. The parameters used for minimal support and whether a patterns is an emerging pattern will be discussed in section 3. The naive method to compute the desired patterns is to compute all subtrees, and select from these previously computed patterns the ones that are frequent. However, this is unfeasible from a computational point of view; the number of subtrees of a tree T is exponential in the size of T. To compute all the frequent patterns, the anti-monotonicity property of the support function ψ is used: $T_i \preceq T_j \Rightarrow \psi(T_i) \geq \psi(T_j)$. With this observation infrequent patterns can be pruned, which reduces the search space drastically.

2.2 FAT-Miner

The main idea for the attribute tree mining algorithm is to split the mining process in a global mining stage and a local one. Loosely speaking the global

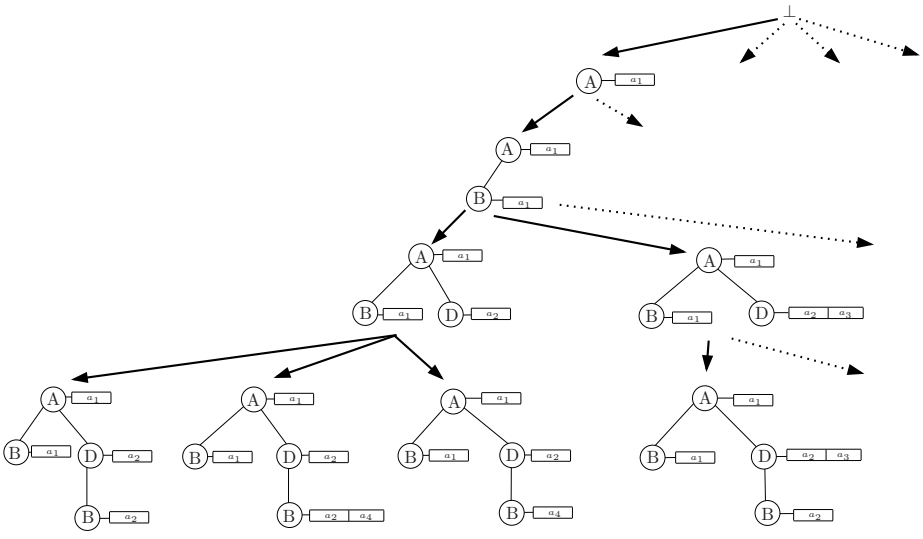

Fig. 2. Part of the enumeration tree for patterns on the tree d_1 (shown in figure 1)

mining part consist of a slightly modified rooted tree mining algorithm as described in the work of Asai et. al [2]: Enumeration of all frequent subtrees is accomplished by using the rightmost extension technique, that is, a $(k-1)$-tree is expanded to a k-tree by adding a new node *only* to a node on the rightmost branch of the $(k-1)$-tree. The local mining must be performed for every node of the subtrees. It boils down to the computation of frequent attribute sets from all the attributes to which the node of the subtree is mapped in the database. In this setting, the local mining algorithm is slightly different from existing frequent itemset mining algorithm [1,3]. This difference is mainly due the fact that the node labels can occur multiple times in a tree. The local and global mining methods have to be combined: the idea is that whenever a candidate tree T (of size k) is generated, the local mining algorithm determines the first frequent attribute set of v_k, say A_1. If there is none, then T must be pruned. Otherwise all supertrees of T are computed, where the attribute set of v_k equals A_1. When there are no more frequent extensions of T left, the next frequent itemset of v_k is computed and this frequent pattern is then further extended. For both the global and local mining a depth-first search through the enumeration lattice is used. In figure 2, this principle is illustrated: the partial enumeration tree of d_1 is shown.

In figure 3 the pseudo code for the local mining algorithm is given. In the function LocMine in case of a non-empty attribute set X, we represent with a_k the attribute which comes last in the ordering of the attributes in X. Recall that we earlier defined that the set of attributes has as element with the highest order a_n. First, for every possible extension of X with a higher ordered attribute than a_k, the frequency is determined. If one of these extensions is frequent,

```
Function LocMine ( X, OCL)
    if X = ∅
    then
        l ← 1
    else
        l ← k + 1
    do
        while l ≤ n
            if support(X ∪ a_l) ≥ minsup
            then
                (X ← X ∪ {a_l})
                return(X,ComputeOcc(X, OCL))
            else
                l ← l + 1
        Y ← X
        if X ≠ ∅
        then
            X ← X \ {a_k}
            l ← k + 1
    while Y ≠ ∅
return (∅, ∅)
```

```
Function ComputeOcc( X, OCL)
    out ← ∅
    for each d_i ∈ OCL
        for each Φ^j_{d_i} ∈ d_i
        if X ⊆ M(Φ^j_{d_i}(v_{k+1}))
        then
            out ← out ∪ Φ^j_{d_i}
return out
```

Fig. 3. The local mining algorithm

```
Algorithm FAT-miner(database D)
    out ← ∅
    C_1 ← candidate one patterns
    for each T ∈ C_1
        out ← out∪ Expand-Trees(T, D)
return out
```

```
Function Expand-Trees(T, D)
    out ← ∅
    do
        (Aset, Nocc) ← LocMine(M(T), occ(T, D))
        if |Nocc| ≥ minsup
        then
            M(v_{k+1}) ← Aset
            occ(T, D) ← Nocc
            out ← out ∪ T
            C_{k+1} ← candidates generated from T
            for each c_{k+1} ∈ C_{k+1}
                out ← out∪ Expand-Trees (c_{k+1})
    while |Nocc| ≥ minsup
return out
```

Fig. 4. The global mining algorithm

the function ComputeOcc is called. This function determines all mappings in a list (occurrence list), for which the rightmost node of the mapping covers the frequent extension. If none of the previous extensions is frequent, a_k is removed from X and X is again extended with attributes.

In the global mining algorithm, as described in figure 4, candidate trees are generated by adding a node on the rightmost path of the tree. For each candidate

one-pattern the function Expand-Trees is called. This function first calls the local mining function, which determines the next frequent attribute set. If there is one, this attribute set is assigned to the right-most node of the current tree, and the result is added to the solution. Then the occurrence list is updated and this tree, with the updated occurrence list, is further extended. Otherwise the current tree is pruned.

3 XML Document Classification with FAT-CAT

The global procedure for the classification of XML documents is as follows:

1. Compute the frequent patterns for the different classes on the training set.
2. Select from these frequent patterns the emerging patterns.
3. The emerging patterns are used to learn the classification model on the training set.
4. Evaluate the classification model on the test set.

To compute the frequent patterns we still have to determine the minimum support value. This, must be done very precisely: when it is set too high, only patterns that are already 'common knowledge' will be found. On the other hand, if it is set too low we have to examine a massive number of patterns if they are useful for the classification task. When dealing with multiple classes in frequent pattern mining, the question arises whether a single support value is sufficient for all different parts of the database. Given two parts D_{c_i} and D_{c_j}, it may be the case that the structural similarity in one part is much higher then the structural similarity in the other part. Hence, in the part with the higher structural similarity a higher support value is preferred. Therefore we have chosen to use k different minimum support values; one for each class label. To determine an appropriate minimum support value for each class, we started with a high support value, and lowered it gradually until a sufficient number of high quality patterns was produced. As sufficient criterion we used that there were at least ten different frequent patterns T of which each has the following property: $\psi(T|c_i)/\psi(T) \geq 0.6$, i.e. the probability of a particular class given pattern T must be greater then or equal to 0.6. This property is also known as the confidence of the rule $T \rightarrow$ class. When this procedure for finding proper support values was applied to the training set, for 53 out of 60 class labels we founded a sufficient number of high quality frequent patterns; these 53 classes together contained roughly about 99% of all XML documents in the dataset.

Having settled the support values for all classes, we still need to select 'interesting' patterns. Since the overall goal is to classify XML documents, we are more specifically interested in patterns that describe local properties of particular groups in the data. Notice that not all computed frequent patterns have this property. These interesting patterns are often called discriminative or emerging patterns [8], and are defined as patterns whose support increases significantly from one class to another. For emerging patterns to be significant, different measures are used. One measure of interest is by what factor observing a

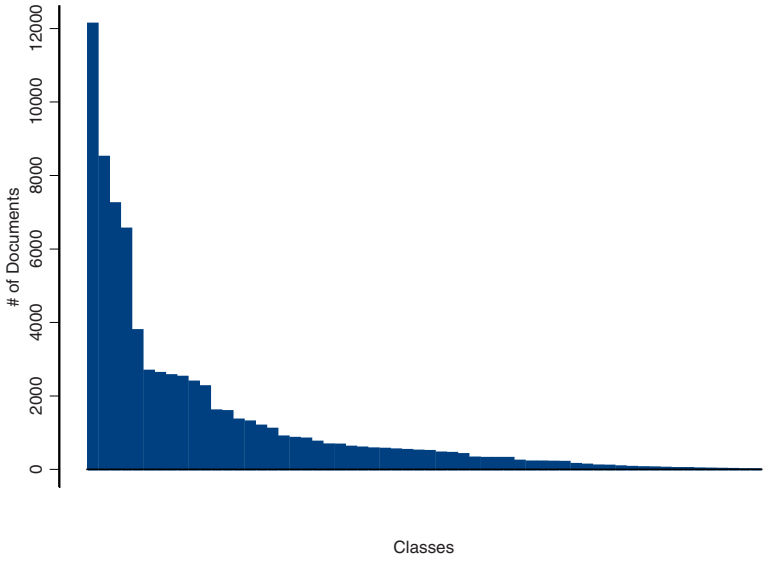

Fig. 5. Distribution of the XML documents over the different classes, classes are sorted according to the number of documents they contain

pattern changes the class probability, compared to the class prior probability, i.e., $P(\text{class}|T)/P(\text{class})$ also known as the lift of the rule $T \rightarrow$ class. Another commonly used measure is the confidence of the rule $T \rightarrow$ class. However, this measure is too tight for the purpose of multi-class classification. This is mainly because the patterns produced would jointly only cover a relatively small part of the records. In this work we used the lift measure. As a result the emerging patterns on the training set covered 99% of the records in the test set; compared with only 51% coverage when the confidence measure was used.

In order to learn a classification model, the next step is to construct binary features for every XML document. Each feature indicates the presence or absence of an emerging pattern in a record. We used decision trees [12] to learn a classification model, more specifically the implementation provided by Borgelt [4]. This implementation uses a top down divide and conquer technique to build the decision tree. At each node in the decision tree, a number of attributes is selected—in a greedy manner—that best discriminates between the classes. As a selection criterion, information gain was used. Additionally, after construction of the decision tree we used confidence level pruning to reduce over fitting. The parameter used with confidence level pruning was set to 50%.

4 Experimental Results

Besides the evaluation of the classification model on XML documents, we also experimentally investigate whether the inclusions of attribute values improves

the performance of the classifier. For this purpose, we trained two classifiers following the procedure described earlier: one where we used the tree structure and the attributes associated to the nodes of the tree, and a second one where only the tree structure was used. These classifiers were trained and evaluated on the Wikipedia XML dataset [7] as provided for the document mining [6] track at INEX 2006. The collection consists of a training set and a test set, which both contain 75,047 XML documents with 60 different class labels. The distribution of the classes over the test set is shown in figure 5. The classifiers were trained on a random sample of approximately one third of the original training set. A sample was taken such that the mining algorithms used were able to run this dataset on a desktop computer with 500MB of main memory. Besides the memory constraint, the running time for the tree mining algorithms already took quite some time (about a week).

The sample used consisted of 25,127 trees and 8,310 distinct node labels; of these nodes 389 contained attributes. To model XML data as attribute trees, a dummy attribute was assigned to each node in the database that had no attributes. The average number of nodes in the trees equals 85, with each attribute counted as a single node, the average size of the trees was 128.

	ATR	NOATR
Micro-average F1	0.479802	0.338321
Macro-average F1	0.527043	0.338920

Fig. 6. Micro-average F1 and Macro-average F1 classification results on the test set

4.1 Results

The document mining track evaluation measures are precision, recall, micro-average F1 and macro-average F1. Macro-average F1 gives equal weight to each class, while micro-average F1 is a per document measure, so it is heavy influenced by larger classes. Furthermore, the $F1$ measure is the harmonic mean of precision and recall: $\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$. We use ATR as abbreviation of the classifier in which we used the tree structure and the attributes associated to nodes, likewise NOATR is the classifier in which we only used the tree structure. The ATR classifier was constructed with the usage of 4,762 distinct binary features of which 1966 were used in the resulting decision tree. Likewise for the NOATR classifier, these numbers were respectively 5,267 and 2,595. The macro and micro-average F1 scores on the test set are shown in figure 6. As expected the scores for the ATR classifier are substantially higher then the scores for NOATR. A closer investigation of the patterns used for the ATR classifier, revealed that the main advantage of including attribute values is that these values often describe the document to which a link points. For example in figure 7 and figure 8 example emerging patterns are shown. However, due to the very common structural properties of these patterns, only the attribute values insures that these are emerging patterns. In the example shown in figure 7, the

attribute value points to a Wikipedia page listing all record labels. In the second example shown in figure 8, the attribute value points to a picture that was the US Navy Jack for some time. Clearly, both values are good indicators of their classes ("Portal:Music/Categories" and "Portal:War/Categories").

```
<section>
<title>See also</title>
        <normallist>
                <item>
                        <collectionlink xlink:type="simple" xlink:href="193135.xml">List of record labels</collectionlink>
                </item>
        </normallist>
</section>
```

Fig. 7. An emerging pattern found on the training set (for clarity displayed with text). This pattern describes class 1474144 ("Portal:Music/Categories") and has a support of 402 in its class and 0 for all other classes.

```
<figure>
        <image xlink:type="simple" xlink:href="../pictures/USN-Jack.png" </image>
</figure>
```

Fig. 8. An emerging pattern found on the training set. This pattern, describing class 2879927 ("Portal:War/Categories") has a support of 48 in its class and 0 for all other classes.

In figure 9 and figure 10 the F1 measure per class are plotted, for the ATR and the NOATR classifier respectively. In both cases the classes were sorted according to the number of documents they contain. When comparing the performance of the classifiers per class, it is noteworthy that the best performance is achieved by classes of moderate size; for the classes that contained very few documents (the smallest six classes) both the classifiers were not able to retrieve any document at all. Furthermore, it is interesting to note the large different in performance between different classes, for example: the ATR classifier for class 148035 ("Portal:Art/Categories") achieved an F1 score of 0.2853 while, for class 2257163 ("Portal:Pornography/Categories") an F1 score of 0.9345 was achieved. In order to accomplish a higher performance for the first class, we experimented with lowering the support value for the class. Unfortunately, the additional patterns did not result in better performance. This suggests, that in the current framework, using only the structure of XML documents is insufficient for classification for all used classes.

Looking at the difference in F1 score per class for ATR and NOATR classifier(shown in figure 11), the ATR classifier substantially outer performs the NOATR classifier for almost every class. However, in two classes the score for the ATR classifier was lower than the score for the NOATR classifier. Hence, for

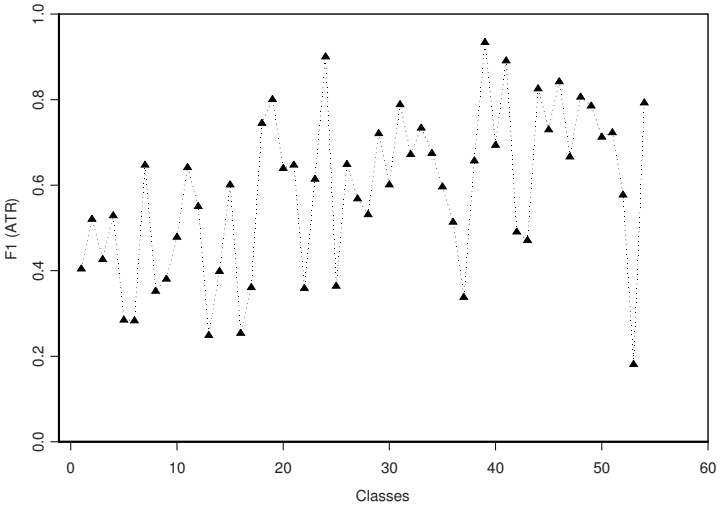

Fig. 9. The F1 score per class, for the experiments where we used the attributes of the data

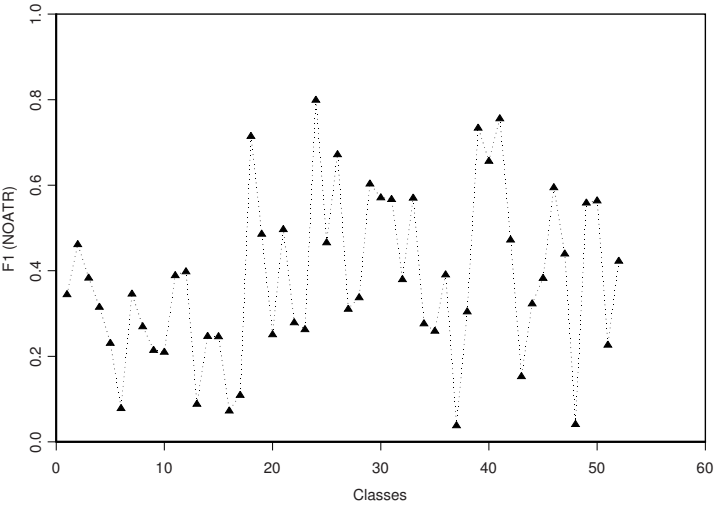

Fig. 10. The F1 score per class, for the experiments where the attributes of the data were left out

some classes the inclusion of attributes has a negative influence on the classification performance. A closer investigation of the emerging patterns for this class is needed to give a possible explanation.

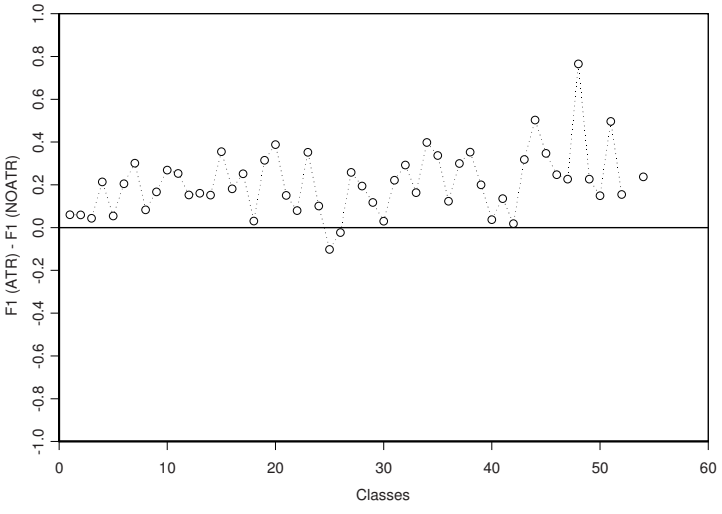

Fig. 11. The difference between the F1 score per class for the ATR classifier and the NOATR classifier. A positive value, corresponds to a higher F1 score for the ATR classifier compared with the NOATR classifier and vice versa.

5 Conclusion

In this work we presented FAT-CAT; an XML classification approach based on frequent attribute trees, and compared these with a frequent tree approach. We have shown that the inclusion of attributes generally greatly improves the performance of the classifier. Furthermore, we analyzed the added value of including attributes into the classification process and describe weaknesses of the used approach.

Further research includes the combination of the current classification approach with more context oriented classification techniques, such as text mining.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pp. 487–499 (1994)
2. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: SIAM Symposium on Discrete Algorithms (2002)
3. Bayardo, R.: Efficiently mining long patterns from databases. In: Laura, A. T., Haas, M. (eds.) SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, pp. 85–93 (1998)
4. Borgelt, C.: A decision tree plug-in for dataengine. In: Proc. 6th European Congress on Intelligent Techniques and Soft Computing (1998)

5. Bringmann, B., Zimmermann, A.: Tree^2 - decision trees for tree structured data. In: European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 46–58, (2005)
6. Denoyer, L., Gallinari, P.: Report on the xml mining track at inex 2005 and inex 2006. In: proceedings of INEX (2006)
7. Denoyer, L., Gallinari, P.: The Wikipedia XML Corpus. SIGIR Forum (2006)
8. Dong, G., Li, J.: Efficient mining of emerging patterns: Discovering trends and differences. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 43–52 (1999)
9. Geamsakul, W., Yoshida, T., Ohara, K., Motoda, H., Yokoi, H., Takabayashi, K.: Constructing a decision tree for graph-structured data and its applications. Fundamenta Informaticae. 66(1-2), 131–160 (2005)
10. De Knijf, J.: FAT-miner: Mining frequent attribute trees. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing to appear (2007)
11. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 80–86 (1998)
12. Ross, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
13. Wang, K., Liu, H.: Discovering structural association of semistructured data. Knowledge and Data. Engineering 12(2), 353–371 (2000)
14. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80 (2002)
15. Zaki, M.J., Aggarwal, C.C.: Xrules: an effective structural classifier for XML data. In: Getoor, L., Senator, T. E., Domingos, P., Faloutsos, C. (eds.) ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 316–325 (2003)